**Inventory Management System:**

This Inventory management system is designed for a warehouse where in the data for the following can be added through Postman:

1. Products
2. Product Category
3. Customer
4. Order Status
5. Place an order using cart
6. Inventory.

**Customer:**

Customer information is required to be in database before placing an order. It is not required when the order is in progress. The information is stored in the database upon entering through Postman.

The following routes work for the customer.

URI: /customer

Method: GET

Request Body: None

Response Body: List<Customer>

URI: /customer/email/{emailId}

Method: GET

Request Body: None

Response Body: Customer

URI: /customer/id/{id}

Method: GET

Request Body: None

Response Body: Customer

URI: /customer/id/{id}

Method: DELETE

Request Body: None

Response Body: None

**Product Category:**

Each product is associated with Product category so this is created before any product is added. A product category can be associated with multiple products. The following routes work for the Product category

URI: **/productcategory"**

Method: POST

Request Body: ProductCategory (just name and no id)

Response Body: ProductCategory

URI: **/productcategory"**

Method: GET

Request Body: None

Response Body: LIST< ProductCategory>

URI: **/productcategory /id/{id}**

**"**

Method: GET

Request Body: None

Response Body: ProductCategory

URI: **/productcategory /id/{id}**

Method: DELETE

Request Body: None

Response Body: None

**Product:**

Every product is associated with product category. This product is shared between inventory & Product category. Following routes work with Product.

URI: **/product**

Method: GET

Request Body: None

Response Body: LIST< Product>

URI: **/product/ productCategory /{productCategoryname}**

Method: GET

Request Body: None

Response Body: LIST< Product>

URI: **/product /productCategory/aboveRating4/{productCategoryname}**

Method: GET

Request Body: None

Response Body: LIST< Product>

URI: **/product/sort/{sortorder}**

Method: GET

Request Body: NONE

Response Body: LIST< Product>

URI: **/product /rating/id/{id}**

Method: PUT

Request Body: Product

Response Body: Product

**Inventory:**

When a product is added, corresponding Inventory item should be added to maintain the inventory of the Product. Following routes work for the Inventory.

URI: **/ inventory /product/quantity/{quantity}**

Method: POST

Request Body: Product

Response Body: Inventory

URI: **/ inventory /id/{id}/quantity/{quantity}**

Method: PUT

Request Body: None

Response Body: Inventory

**Order status:**

It holds 3 rows: In progress, place order and ordered.

Placing an order using cart:

Cart can contain multiple products ids and the corresponding required quantity.

When a cart is received, the cart id is null so it is considered a new order and it sets the order status as In progress.

During In progress, products can be removed / added or the item quantity can updated.

When the order status changes to place order, inventory is checked if the items are available and the order is placed and the status is changed to Ordered.

Following api routes work for the order status.

URI: **/orderstatus**

Method: GET

Request Body: None

Response Body: List<OrderStatus>

URI: **/orderstatus**

Method: POST

Request Body: String

Response Body: OrderStatus

URI: **/orderstatus /{id}**

Method: DELETE

Request Body: OrderStatus

Response Body: String

**CART:**

When a client wants to place the order, there is provision for adding multiple product Id’s and the corresponding quantity to the cart. Sample cart is shown below. If the quantity asked is not in the inventory, an exception is thrown.

If the client wants to add / remove product from the cart or to update the quantity, it can be done as well as long the order status is “In Progress”. When the cart is added to the postman at first, the cart Id is null and the order status is also null. Upon executing, the application gives the cart the id and the order status is “In progress”. Inventory checking happens upon all operations.

When it is time to place order, the client changes the status to “Place Order”, the inventory is checked, if the cart id is already available in the database, product id validation is done and inventory updating is also done. Finally the total is calculated and the cart is added to the database with status “Ordered”

Client can also view the Order history at any stage.

Sample cart that client uses to create an order:

{

"cartId": null,

"customer": {

"customerId": 5,

"customerName": "Sandy",

"customerAddress": "123 main st, plano, tx",

"customerEmailId": "sandy@gm.com"

},

"products": [

{

"productId": 2,

"quantity":1

}

],

"orderStatus": null,

"cartTotal": 10.0

}

Following routes works for the Cart / Order history.

URI: **/order**

Method: GET

Request Body: NONE

Response Body: List<Cart>

URI: **/order**

Method: POST

Request Body: Cart

Response Body: Cart

URI: **/order /customer/{customerid}**

Method: GET

Request Body: None

Response Body: List<Cart>

Following requirements have been met.

1. The inventory management system must account for multiple different product categories such as beds, televisions, sofas, shoes, etc.

A product category was designed to accommodate all different categories

1. Each product must be associated with a category

This is done using @ManyToOne relationship

1. Each product should have a price, quantity, and rating (1-5)

Product has price and rating with values allowed are 0,1,2,3,4,5. The Quantity is separated from the product to the inventory Table. Join was used for this purpose

1. System should store previous orders.

When an order is placed or the order is in progress, the order is added to the database and user can view it with the api route mentioned under Cart section

1. Retrieve all products in a specified category

Since both product and product category are joined using @ManyToOne and @OneToMany, all the products can be viewed under a specified category id

1. Retrieve all products in a specified category above a rating of 4.

This is done with the above said rule and the data was filtered for ratings above 4

1. Be able to sort products based on price in ascending or descending order. This sort must be done using your own algorithm.

Lambdas and streams were used to sort the products based on asc /desc order. It also throws an exception when sort order than asc / desc is specified.

1. Ability to create a cart that will sum the total price of all items in the cart

When the order is ready to be placed, the application calculates the total price for all the items in the cart.

1. Ability to add/remove products to/from the cart.

The application handles when the items in the cart are added / updated with new quantities / removed from the cart.

1. Ability to create new categories

API route for handling this requirement is mentioned under Product category section

1. Ability to create new products

New Products can be added under the product category that is already present in the database

1. Ability to add/update quantity based on items in the cart

The inventory gets updated based on the items in the cart when the order is ready to be placed

1. Ability to add/edit ratings to an existing product

Ratings to a product can be updated between 0 and 5. API route is specified under product section.

1. Ability to retrieve order history.

Application can display all the orders that are placed or the orders that are in progress state. API route is mentioned under Cart section.

1. This must allow the client to see all items and quantities of each order.

API route is mentioned under Cart section wherein the orders can be displayed from the order history for the customer id supplied in the api route.